

How to Build a Docker Container for Raspberry Pi in the Cloud.

Raspberry Pi用Dockerコンテナをクラウドでビルドする



DAY

2019/May/24

COMPANY

SAKURA Internet Inc.

DEPARTMENT

SAKURA Internet
Research Center.

NAME

Senior Researcher
KIKUCHI Shunsuke
菊地 俊介

Part 0.

Motivation, Background. / なにがやりたいのか

Part 1.

How to use Container on Raspberry Pi.

Raspberry Piでコンテナを動かす

Part 2.

How to build Container for Raspberry Pi in the cloud.

Raspberry Piのコンテナをクラウドでビルドする

菊地 俊介 (東京都出身、品川区在住)



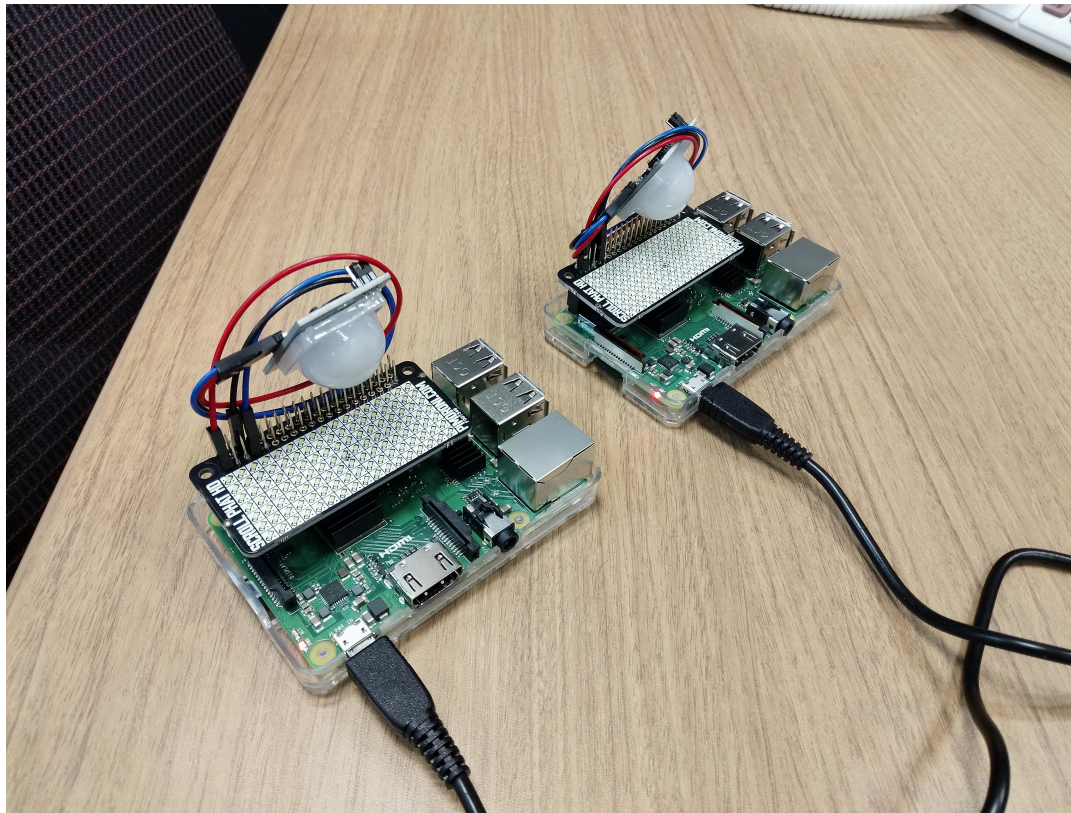
- 所属** さくらインターネット研究所
- 経歴** 早稲田大学大学院 卒
富士通（株）富士通研究所
ネットの研究やったり、トイレIoT作ったり
さくらインターネット研究所
データ流通実証実験、Fogコンピューティング、
AR/VR、量子（アニーリング）コンピュータ
- 専門** エッジ・Fogコンピューティング
(分散系システムのあたり)
- 趣味** 新技術調査、家庭内IoT、車、鉄道

Part 0.

Motivation / なにがやりたいのか

- IoTやそれを発展させたサイバーフィジカル・システムによる、**実世界（現場）の高度化**が今ホット。
- 本業として、コンピューティングリソースが現場に溶け込んでいく（近）未来の実現にむけて、**エッジコンピューティング・Fogコンピューティング**を研究開発。
- IoT、Fogコンピューティングの実装に今一番便利なのは**Raspberry Pi**。

- 最もお手軽なIoTマシン。
 - GPIOが使えて、Linuxが動く。
 - すでにエコシステム（豊富な3rdパーティ部品、開発ノウハウ）が確立。



さくらインターネット研究所
で実施中の、
Raspberry Piを用いた
Fogコンピューティング試作

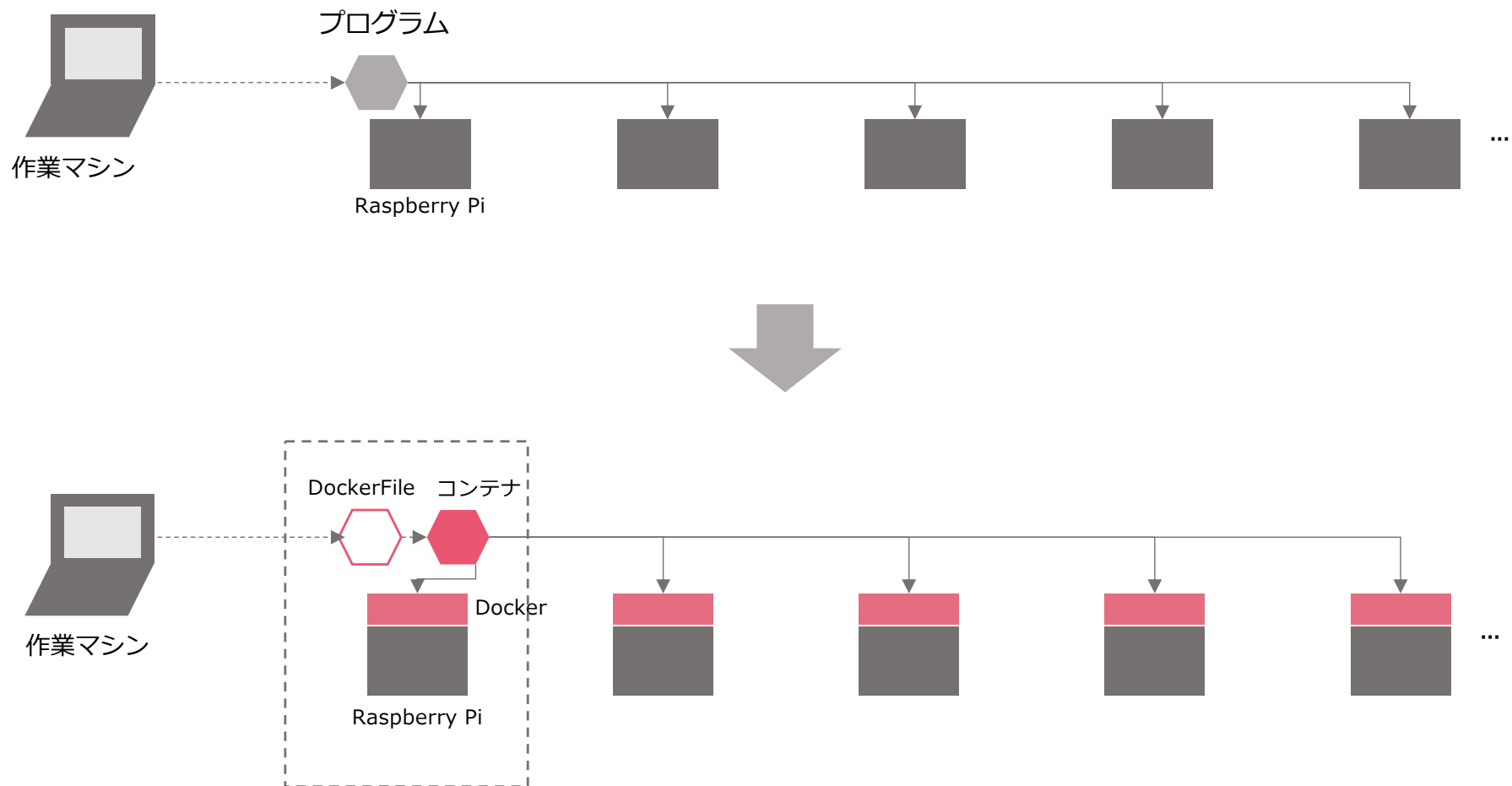
- **たくさんのRaspberry Piを並べてFogシステムを作りたい。**

- 台数が増えてくると、個々のRaspberry Piに対して作業をしていられなくなる。
 - →デプロイシステムがほしい
- 台数が増えてくると、個々のRaspberry Piに対して状態管理をしていられなくなる。
 - →マネジメントシステムがほしい
- **Raspberry Piとコンテナエコシステム（Docker, Docker-compose, Kubernetes）を組み合わせて使いたい。システムの開発の高度化（CI/CD）を実現したい。**

Part 1.

Raspberry Piでコンテナを動かす How to use Container on Raspberry Pi.

- Raspberry Piでもコンテナベースの開発に移行したい



- →できます。しかもそれほど難しくなく。

- RaspbianOSに（普通に）Docker入ります。
 - ただし、Raspberry Pi Zero(WH)もターゲットにする場合は version=18.06.3まで。
 - それ以降だとZeroでインストールできなくなる。

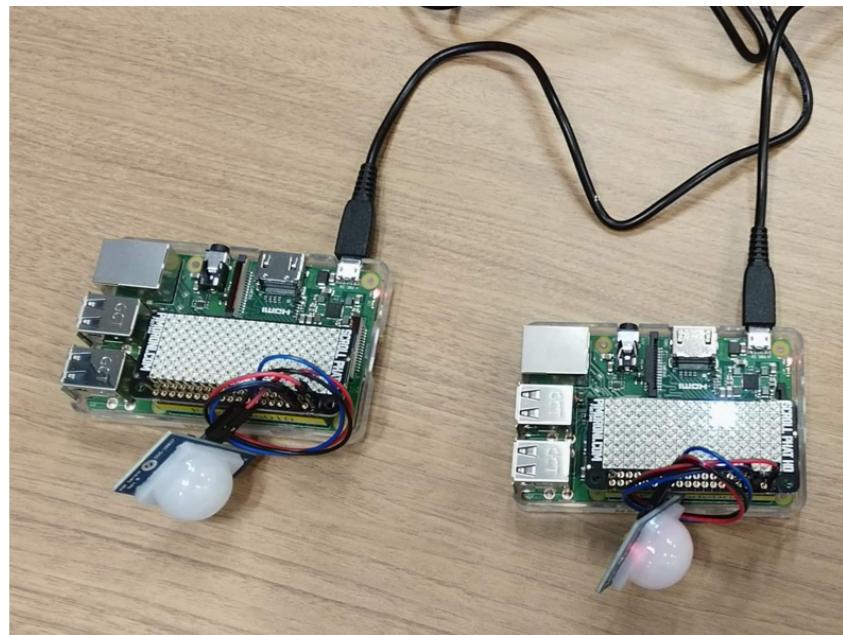
```
$ sudo apt-get update  
$ sudo apt-get install curl
```

```
$ curl -sSL https://get.docker.com | VERSION=18.06.3 sh
```

- Docker-Composeは、（入るけど）インストールが難しい
 - 本家からはRaspberry Pi(arm7l)用バイナリが提供されていない。
 - 自前でビルド必要だがエラーで止まる等、一筋縄ではいかない。
 - pipで入れるのが楽。
 - <https://qiita.com/Amb98/items/23b455a83c8a6dacao5b>
- Kubernetesも使えます。
 - k3sで動作確認できた。（本発表ではスコープ外とします）

- コンテナ使えれば楽なのはわかっているのだけど、コンテナじゃRaspberry Pi固有のハードウェアにアクセスできないのでは...?
 - →ファイルシステムとして見えているハードウェアであれば、ホストデバイスへのアクセスを指定することで、扱えます。

```
$ docker run --name app --device /dev/gpiomem --device /dev/i2c-1 -d app
```

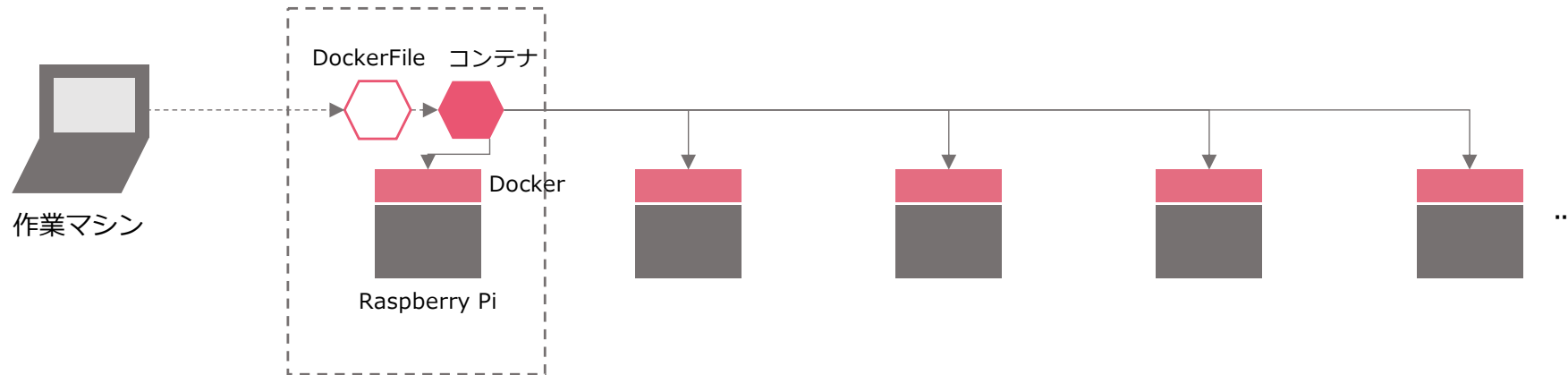


コンテナのアプリから、
GPIO（焦電センサ）と
i2cデバイス(LEDアレイ)に
アクセスしている様子

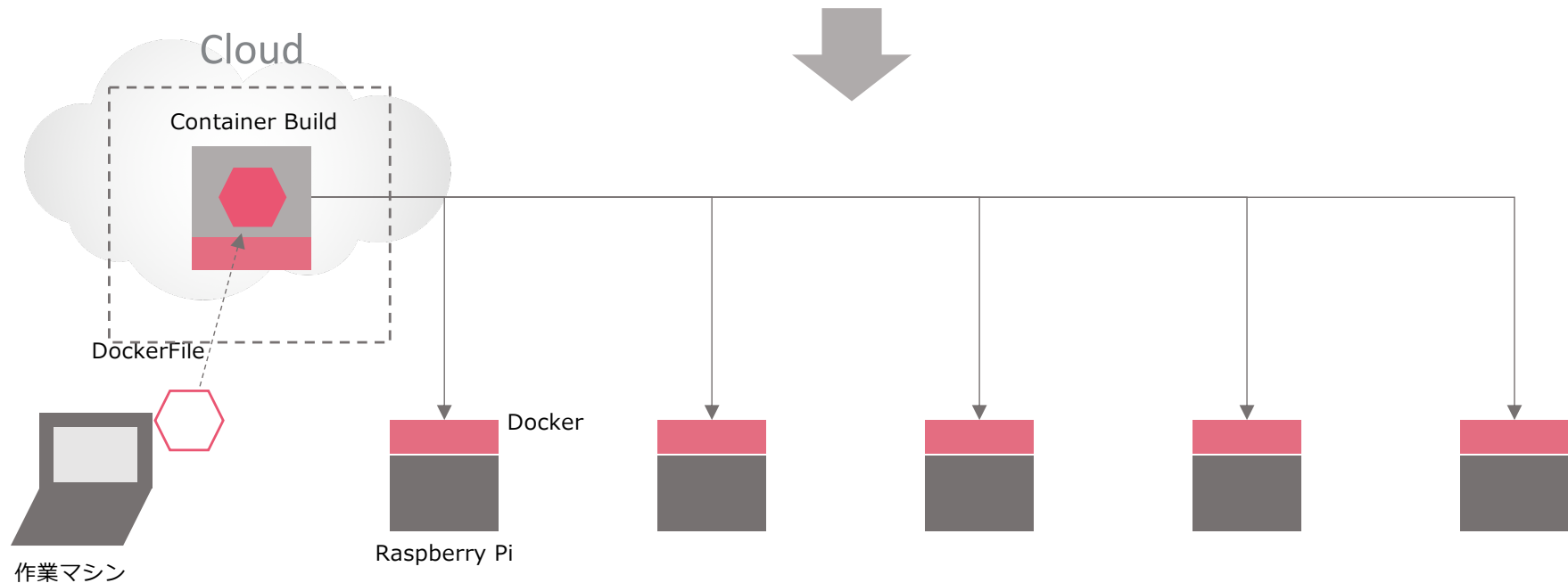
Part 2.

Raspberry Piのコンテナをクラウドでビルドする
How to build Container for Raspberry Pi
in the Cloud.

- Raspberry Piではコンテナイメージビルドが遅い...
 - ちょっとアプリを作り込み始めると、buildに30分とかざら。



Idea : クラウド上でコンテナイメージビルドできれば早くなるんじゃ...?

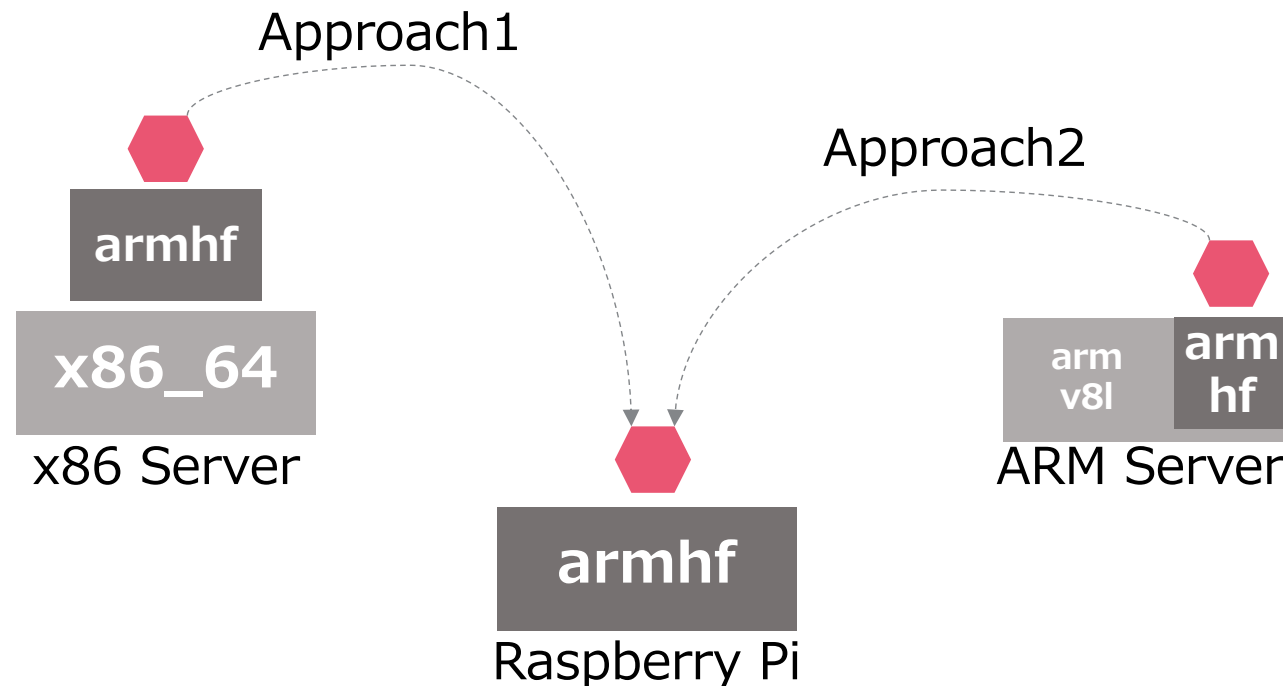


- 一見簡単そうに思えるが、実はそうでもなかった。
 - Raspberry PiはARMプロセッサ搭載だから、同じARM採用のサーバでイメージビルドすればOK ?!
 - Amazon EC2 A1インスタンス、Packetクラウド ARMベアメタルサーバなどARMサーバのクラウドも出始めているので。
- 難しさの原因
 - Raspberry Pi 3B+は64bit ARMプロセッサ搭載
 - Raspberry Pi Zeroなどは、32bit ARMプロセッサ搭載
 - 下位互換性確保のため？か、**RaspbianOSは32bitOS**として作られている
 - →**Raspberry Pi用イメージは32bit(armhf/armv7l)**で作成する必要がある
 - 一方、世の中のほとんどの**ARMサーバとOSは、64bit(arm64/armv8l)**への移行を済ませている
 - **ARMサーバ上でさっとイメージビルドしてRaspberry Piに持ってくる、という訳にはいかない**

Special Thanks!! to
Mr. Nishinaga, Mr. Hiramatsu

- 2つのアプローチ

- アプローチ1 : x86サーバ上の`armhf`エミュレーション環境でのイメージビルド
- アプローチ2 : ARMプロセッサ搭載サーバでのバイナリ互換モードを利用したイメージビルド



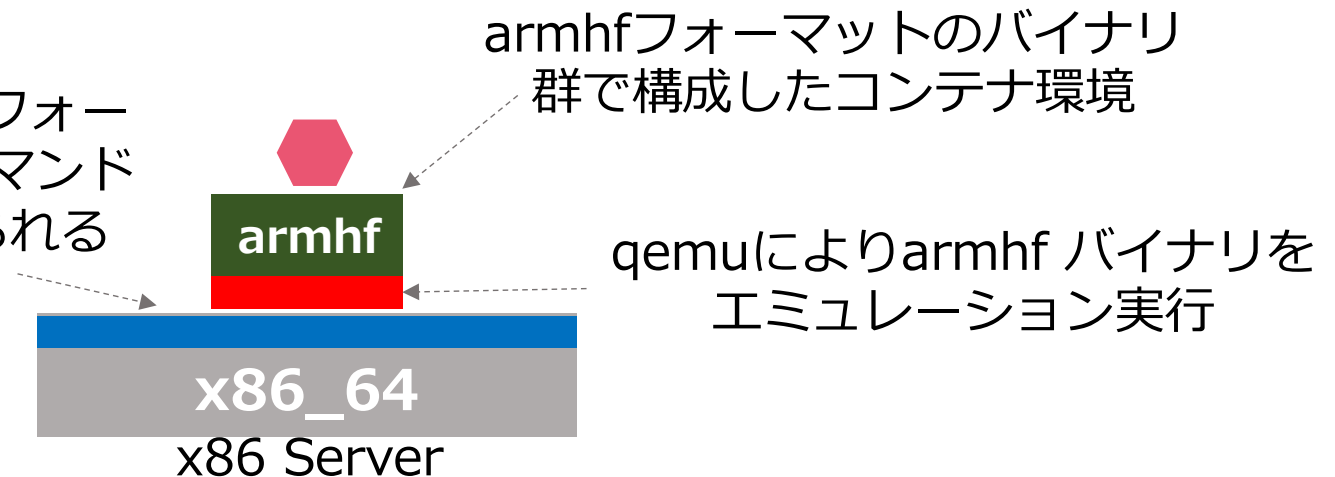
- multi-archプロジェクトのツール・コンテナを利用することで、x86アーキのマシン上で多数のターゲットアーキのバイナリ（コンテナ）を作成する事が可能

- <https://github.com/multiarch>

- 動作原理

- qemu + binfmt-misc + docker
- (= ターゲットアーキエミュレーション + シームレスバイナリ実行 + 環境アイソレーション)

binfmt-miscにより、バイナリフォーマットの種別に応じて任意のコマンド（エミュレータ）を実行させられる



- 実際の使い方例(1/2)

```
[root@fogregistry qemu]# uname -a
Linux fogregistry 3.10.0-957.5.1.el7.x86_64 #1 SMP Fri Feb 1 14:54:57 UTC 2019
x86_64 x86_64 x86_64 GNU/Linux
[root@fogregistry qemu]# docker run --rm --privileged multiarch/qemu-user- ←これだけ！
static:register
...<snip>...
[root@fogregistry qemu]# docker build -t armhf-artful-vim -f Dockerfile .
...<snip>...
[root@fogregistry qemu]# docker run -it --rm armhf-artful-vim:latest bash
root@e8418d14f0e5:/# uname -a
Linux e8418d14f0e5 3.10.0-957.5.1.el7.x86_64 #1 SMP Fri Feb 1 14:54:57 UTC 2019
armv7l armv7l armv7l GNU/Linux
root@e8418d14f0e5:/# file /bin/bash
/bin/bash: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0,
BuildID[sha1]=c2fac9c07846ccd7daa2c96126fa93ce863b2ea4, stripped
root@e8418d14f0e5:/# exit
exit
[root@fogregistry qemu]#
```

- 実際の使い方例(2/2)

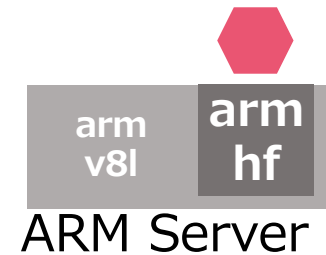
Dockerfileで、ベースになるイメージを指定する

```
FROM multiarch/ubuntu-core:armhf-bionic
```

```
RUN apt-get update && ¥  
    apt-get install -y vim
```

コンテナの内部に、qemu本体が含まれる

```
root@e8418d14f0e5:/# cd /usr/bin  
root@e8418d14f0e5:/usr/bin# ls q*  
qemu-arm-static  
root@e8418d14f0e5:/usr/bin#
```



- 調査したものの、方法を発見できず。
- armv8lマシン上でarmhfのバイナリを実行する方法自体は存在している。
 - <http://arch.jpn.org/archives/272>
- arm32のバイナリを作る方法は、クロスコンパイル環境を構築するしかない？
- その環境をdockerコンテナ内に隔離する方法は？
- 詳しい方のご助力をお願いしたい

- Raspberry Pi 3B+で直接コンテナをつくる場合に比べて、
 - x86サーバ上でarmhfコンテナイメージを作成する場合
 - (ARMv8lサーバ上でarmv8lコンテナイメージを作成する場合、参考情報)

について作成時間を評価した。

計測方法

- docker/docker-composeのコンテナビルド時間をtimeコマンドで計測

評価環境

1. Raspberry Pi 3B+ / Raspbian OS(stretch)
2. x86_64 1コア1GBメモリ、SSD、さくらのクラウド上 / Ubuntu 18.04.3
3. x86_64 2コア1GBメモリ、SSD、さくらのクラウド上 / Ubuntu 18.04.3
4. ARM64 64コア256GBメモリ、HDD(?), HP Apollo 70 System, (ThunderX2) / Ubuntu 18.04

■ ごく簡単なDockerfile

コンテナのターゲットアーキ

評価対象 システム		simple-x86	simple-armhf	simple-arm64
	Raspberry Pi 3B+	NG	5m35.115s	NG
	x86_64 1core	0m31.054s	1m23.923s	1m10.271s
	x86_64 2core	0m28.809s	1m11.900s	1m7.687s
	Apollo70	NG	NG	1m6.642s

■ 典型的アプリ想定Dockerfile

	app-x86	app-armhf	app-arm64
Raspberry Pi 3B+	NG	31m40.088s	NG
x86_64 1core	2m12.228s	39m10.197s	(not yet)
x86_64 2core	1m48.210s	32m47.068s	31m4.249s
Apollo70	NG	NG	14m23.764s

- あまり作り込まず基本的な構造のものを用意した

■ ごく簡単なDockerfile

```
FROM multiarch/ubuntu-core:armhf-bionic
```

```
RUN apt-get update && ¥  
    apt-get install -y vim
```

■ 典型的アプリ想定Dockerfile

```
FROM multiarch/ubuntu-core:armhf-bionic
```

```
WORKDIR /app
```

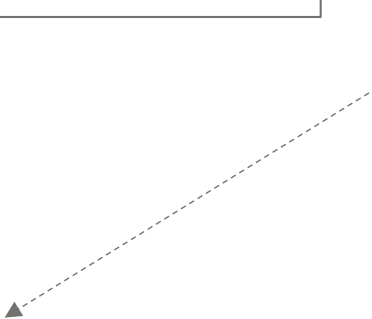
```
COPY requirements.txt .
```

```
RUN set -x ¥  
    && apt-get update ¥  
    && apt-get install -y build-essential  
    python3-smbus python3-pip  
RUN pip3 install -r requirements.txt
```

```
COPY . .
```

```
EXPOSE 8630  
CMD ["python3", "-u", "app.py"]
```

```
Flask==1.0.2  
pyserial==3.4  
RPi.GPIO==0.6.5  
scrollphatd==1.2.1  
smbus==1.1.post2  
requests==2.21.0
```



- Raspberry Pi 3B+ネイティブに対して、x86_64サーバ上でのコンテナビルドの性能は？
- 結果
 - シンプルDockerfileで、5m35.115s→1m23.923s/1m11.900sとなり、およそ1/5に改善
 - 典型的アプリDockerfileでは、31m40.088s→39m10.197s/32m47.068sとなり、わずかなが性能悪化
- 考察
 - x86_86でのビルドの様子を見たところ、numpyのビルドに大変時間がかかっていた。このことから、qemuのエミュレーションでは浮動小数点演算がソフトウェアで遅いのでは？と推測される。

- ARM64サーバ（HP Apollo70）の、x86サーバ（エミュレーション）に対する性能は？
- 結果
 - シンプルDockerfileで、1m6.642s→1m10.271s/1m7.687sとなり、x86のエミュレーションよりわずかに早くなっている。
 - 典型的アプリDockerfileでは、14m23.764s→ 31m4.249sと2倍程度の速さ。
- 考察
 - x86のソフトウェアエミュレーションに対して劇的に早いとは言えない。典型的アプリのケースで2倍早いのは、おそらくx86では浮動小数点演算がソフトウェア処理であることによるボトルネックに起因するのではないか。
 - ARMの64コアを使いこなしている様には思われない。（パラメータチューニング等未確認）

- IoT/Fogコンピューティング用途で、Raspberry Piをたくさん並べて使いたいのので、コンテナ+デプロイ環境を用意したい。そのニーズに答えられるソリューションを検討した。
- Raspberry PiにDocker/Docker-Compose/Kubernetesを入れて使うことはできる。オススメ。
- コンテナビルドをクラウドで高速化する目論見は、仕組みとしては可能だが性能があまり出ないという結果に。アプリ（ソフトウェア）を選ぶ可能性がある。
- ARM64サーバを32bitバイナリ作成には使えないことがわかった。原理的にできないとは考えづらく、追跡調査・識者のご意見求む。
- そもそもARM64サーバの性能が良くない。OSチューニングが必要/待たれる。
- Raspberry Piは、RaspbianOSを捨てるべきかもしれない。

Thank you for Listening !!